2010

Ontology modeling of UBL process diagrams using OWL

Suman Roy
Infosys Technologies Ltd., India

Kiran Prakash Sawant
Infosys Technologies Ltd., India

Aditya K. Ghose
University of Wollongong, aditya@uow.edu.au

Ontology modeling of UBL process diagrams using OWL

Abstract

We present a logical framework for modeling Universal Business Language (UBL) processes. The proposed framework provides a representation of the dynamic world being modeled on the user supplied axioms about preconditions and the initial state of the world, and produces a workflow specification at a higher level of abstraction. We use the Frieghtbilling process of the UBL as a case study for our experiment. Further we extract ontology from the associated UBL document that can ensure efficient information retrieval, discovery and auditing. We use the popular semantic web formalism, Web Ontology Language (OWL) for ontology construction purposes. These domain ontologies can play a useful role in many applications, like constraining requirements.

Disciplines

Physical Sciences and Mathematics

Publication Details

Roy, S., Sawant, K. & Ghose, A. K. (2010). Ontology modeling of UBL process diagrams using OWL. 2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM) (pp. 535-540). Piscataway, New Jersey, USA: IEEE.

Ontology Modeling of UBL Process Diagrams using OWL

Suman Roy[1] Kiran Prakash Sawant[2] Aditya K. Ghose[3]

Abstract—**We present a logical framework for modeling Universal Business Language (UBL) processes. The proposed framework provides a representation of the dynamic world being modeled on the user supplied axioms about preconditions and the initial state of the world, and produces a workflow specification at a higher level of abstraction. We use the Frieghtbilling process of the UBL as a case study for our experiment. Further we extract ontology from the associated UBL document that can ensure efficient information retrieval, discovery and auditing. We use the popular semantic web formalism, Web Ontology Language (OWL) for ontology construction purposes. These domain ontologies can play a useful role in many applications, like constraining requirements.**

Keywords: UBL, UBL process modeling, ontology, OWL, semantic rules.

I. Introduction

In business, one needs to routinely exchange data with trading partners to successfully negotiate and execute transactions. This can be facilitated by re-using standard patterns in documents. Adopting a common standard can reduce development and maintenance costs, improve performance, and enhance business relationships. Universal Business Language (UBL) [20] is an OASIS standard [14] to develop common business document schemas to provide document inter-operability in the electronic business domain. UBL comes with a library of reusable components such as Address, Price and a set of document schemas such as Order, Invoice to be used in e-business. UBL also provides a diagrammatic description of business processes using a notation similar to UML activity diagram with roles. UBL process descriptions along with their interlinked documents are fast becoming popular with public and private sector organizations around the world.

A UBL process flow consists of a collection of cooperating, coordinated activities meant to execute a process. An agent, in the form of a human, a device or a program can perform an activity in a process flow. UBL process flows have lot of similarities with traditional workflows. Researchers have proposed many formal models for analyzing and reasoning about workflows [5], [8], [7], [12]. Frameworks based on graphs, event-condition-action rules, and various logics are widely used for specifying workflows. Visualization of a control flow is possible using graph-based approaches, where nodes are associated with activities and edges with control or data flow between activities. A well-known formalisms that is applied to specify workflow are Petri nets [19]. Event-condition-action rules have been sparingly used in the specification of workflows [9]. However, control flow graphs are more expressive than these formalisms. Logic-based formalisms use the power of declarative semantics of logic to specify the properties of workflows and the operational semantics to model the execution of workflows. In [7], the authors propose such a logic-based framework for the specification and execution of workflows using a logic programming style of language, called event calculus. We believe that for UBL processes, even a decidable fragment of first-order logic will suffice for the specification of control flow graphs, and capturing the execution dependencies between activities and scheduling of activities within a flow. OWL (a variant of Description Logics) with rules seems to be the perfect choice for this work.

Effectively managing the data stored in UBL documents is not an easy task, especially when it comes to ensuring efficient information retrieval, discovery and auditing; the challenge is to extract meaningful information from the large amount of data available. Therefore, it is necessary to add structure and semantics to provide a mechanism to more precisely describe data for these UBL business documents. Further, for semantic information to be useful, it should be able to define characteristics that the document should possess, such as the methods of ordering and payment, constraints on spatial and temporal availability etc. Such semantic information can be defined effectively though ontology languages like, semantic web initiatives [4], OWL [3], and to name a few. We felt that it would be useful if we can use OWL to extract meaningful information out of UBL documents and to relate them in a wider business context. Thus, the ontology extracted from both a UBL process and the linked documents can be merged in a meaningful way to create domain ontology for the corresponding UBL artifact.

II. UBL process details

XML has been adopted in a number of industries as a framework for the definition of the messages exchanged in electronic commerce. The widespread use of XML has led to the development of multiple industry-specific XML versions of such basic documents as purchase orders, shipping notices, and invoices. While industry-specific data formats have several advantages, the existence of different formats to accomplish the same purpose in different business domains also has significant disadvantages. The OASIS Universal Business Language (UBL) [20] is intended to help solve these problems by defining a generic XML interchange format for business documents that can be modified to meet the requirements of particular industries.

UBL comes with a library of reusable components such as Address, Price, and a set of document schema such as Order, Invoice, Remittance Advice; which are meant for use in e-business. UBL is designed to plug directly into existing business, legal, auditing, and records management practices. It is designed to eliminate the re-keying of data in existing fax and paper-based business correspondence and provide an entry point into electronic commerce for small and medium-sized businesses. UBL 2.0 traces its

[1]Software Engineering and Technology Labs (SETLABS), Infosys Technologies Ltd.,
44 Electronics City, Hosur Road, Bangalore 560 100, India
email: Suman_Roy@infosys.com

[2]SETLABS, Infosys Technologies Ltd.
Plot No. 1, Rajiv Gandhi Infotech Park,
Hinjawadi, Taluka Mulshi, Pune 411 057, India
email: KiranPrakash_Sawant@infosys.com

[3]School of IT and Computer Science,
University of Wollongong, NSW 2522 Australia
email: aditya@uow.edu.au

978-1-4244-7818-7/10/$26.00

origins back to the EDI standards and other derived XML standards. In total, there are 31 documents covering business needs in the phases of pre-sale, ordering, delivery, invoicing and payment. These documents and components are designed to support the typical business processes for covering a wide range of supply chain.

III. Specification of UBL process details using a logical framework

As UBL processes are quite similar to UML activity diagrams with roles, specification of such a process model involves capturing relevant aspects of its constituent activities, the relationships among activities and their execution requirements. In a process flow, activities are related to each other through control flow relation/transition relation. It is possible to identify process flow with a few transition patterns [7] such as, *sequential*, *parallel*, *conditional*. *iteration* etc. A sequential flow is where an occurrence of an activity is followed by another activity, or equivalently, an occurrence of an activity can be preceded by occurrence of another activity, i.e., activities are executed in sequence. In a parallel transition, two or more activities are executed in parallel. A conditional transition takes place on a condition, which specifies that one of the alternate activities is executed. All these patterns can be expressed using rules in a formal framework.

Further a process may contain some documents, which may be semi-structured. For certain purposes like requirements editing, it is required to retrieve, discover and audit information hidden in the data of those documents. Therefore, it will be useful to add structure and semantics to provide a mechanism for more precisely describing the data in these documents. Such semantic information can be defined effectively through ontology languages.

A control flow graph associated with a process can be described using logical formulas. For example, the following successor relations can be used: $initiates(.,.), terminates(.,.), follows(.,.)$ etc. A relation $initiates, (terminates)$ will indicate a (possible) pre-condition (post-condition) the process. Similarly, $follows(.,.)$ will capture the consecutiveness of two activities. Also, we use another predicate $happen(e)$ to say that event e has occurred. In this paper we shall restrict our vocabulary to only these predicate symbols, and few others. We admit that this list is no way exhaustive, and we hope to expand it in a future work.

These relations can be suitably defined using predicate relations in a first-order formula. The set of predicates maps the formal structure of the control flow graph directly into a set of formulas of formalisms like, event calculus [7], transaction logic [8], π-calculus [16] and to name a few. One may also note that the documents embedded in a process may be a text, or semi-structured documents in the form of XML. Though it it not possible to associate semantics with XML, people has tried to discover some kind of meaning out of XML documents and map them to other ontological languages like RDF [2], OWL [6]. For the time being, we shall assume that all these formulas/rules are being expressed in a framework of first-order logic (using Horn rules).

IV. Specifying a process using rules

We use a logic programming like formalism to write down specifications for UBL processes. Our approach is similar to the kind of exercise done for specification of workflows using event calculus [7]. Although activities in a process flow may have duration, for ease of specification we shall not take time into account. For each activity a that has started we associate events $start(a)$, and $end(a)$ to denote the beginning and completion of activity a. We mention again that $follows(a_1, a_2)$ expresses that activity a_2 follows a_1. For denoting that activity a_1 makes transition to activity a_2 through conditional gateways or vacuously, we use the role $transitsto(a_1, a_2)$. We can assume some transitive property to hold good in this case: $follows(a_1, a_3) \longleftarrow follows(a_1, a_2), follows(a_2, a_3)$.

We use few axioms to state that a property holds under certain conditions. Also note that we assume all the events are instantaneous, there is no time spent from the beginning to the end of an event.

$$holds\_at(p) \longleftarrow happens(e), initiates(e, p),$$
$$not\ interrupted(p)$$
$$holds\_at(p) \longleftarrow happens(e), terminates(e, p),$$
$$not\ interrupted(p)$$
$$interrupted(p) \longleftarrow happens(e'), terminates(e', p)$$

While the predicate $holds\_at(p)$ denotes that the property p holds, predicate $happens(e)$ says that the event has occurred. The explanation of the rest of the predicates used are in order. $initiates(e, p)$ denotes that the event e initiates a period of time (implicit) during which the property p holds, and $terminates(e, p)$ says that event e puts an end to a period during which p was true, $interrupted(p)$ represents that property p ceases to hold. The *not* operator is interpreted as "negation-as-failure".

Sequential activities

Let us now try to specify sequential activities in OWL. Suppose the activity a_j can start unconditionally, when activity a_i finishes. This is captured as (see Figure 1):

$$transitsto(a_i, a_j) \longleftarrow follows(a_i, a_j), happens(end(a_i))$$

We can define an inverse property too. Defining an inverse role $before(a_j, a_i)$ corresponding to $follows(\cdot, \cdot)$ we introduce the following rule.

$$reversesto(a_j, a_i) \longleftarrow before(a_j, a_i), happens(start(a_j))$$

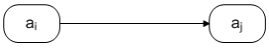

Fig. 1. Sequential Activity

Concurrent activities

In a process flow, some activities might be executed concurrently, specifically activities after an AND-split are scheduled to be executed concurrently. Figure 2(a) shows an AND-split. Activities $a_1, a_2, \ldots, a_n$ can start only when the activity $a_j, j \notin \{1, 2, \ldots, n\}$ finishes, and those activities occur concurrently. This can be captured as follows.

$$transitsto(a_j, x) \longleftarrow and\_split(a_j, L), happens(end(a_j)), member(x, L)$$

where $L = [a_1, \ldots, a_n]$ is a list of actions. Note that the variable x assumes value from the set $\{a_1, \ldots, a_n\}$. We

have used a predicate $and\_split(a_j, L)$ to denote that activity a_j is split into a list L of activities, and predicate $member(xL)$ to denote that variable x for an activity is a member of list L.

In an AND-join (Figure 2(b)) the activity a_j can start when all the preceding activities $a_1, \ldots, a_n, j \notin \{1, \ldots, n\}$ finish. These activities may not be completed concurrently, however, we do not take time of their completion in our model. The following rule can be used to represent the execution of an AND-join.

$$transitsto(x, a_j) \longleftarrow and\_join(L, a_j),$$
$$happens(end(a_1)), \ldots, happens(end(a_n)), member(x, L)$$

The predicate $and\_join(L, a_j)$ indicates that the activities in the list L are merged into the activity a_j. Further, we add the facts, $member(a_1, L) \cdots, member(a_n, L)$ etc.

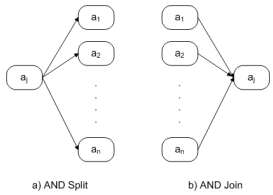

Fig. 2. (a) AND-split and (b) AND-join

Conditional activities in a XOR-gateway

In a process some of the activities get enabled depending on certain conditions, otherwise they are not executed. The important point to notice here is that only one of the conditions should hold at the time of decision, so that only one path is taken.

In an XOR-split (see Figure 3(a)), when activity a_j finishes, one of the activities $a_1, \ldots, a_n$ (assume $i \notin \{1, \ldots, n\}$) can begin depending on whether the condition associated with that particular activity is satisfied. The condition may be a state check (*i.e.*, $holds\_at(.)$ predicate) The conditions on transitions are mutually exclusive. Suppose on the completion of the activity a_i, one of the activities among $a_1, a_2, \ldots, a_n$ can be executed depending on the condition evaluated. This can be specified as follows:

$$transitsto(a_j, a_2) \longleftarrow xor\_split(a_j, L), happens(end(a_j)),$$
$$pair(a_2, cond_2), holds\_at(cond_2), belongsto(cond_2, condList)$$

Here predicate $xor\_split(a_j, L)$ denotes that the activity a_j gets split into a set of activities $a_1, \ldots, a_n$, $belongsto(cond_2, condList)$ denotes that $cond_2$ is in the set of conditions $condList$ associated with the particular XOR-split. Simultaneously, we add the roles, $pair(a_1, cond_1), \ldots, pair(a_n, cond_n)$, where $pair(a_i, cond_i)$ has the obvious meaning that activity a_i is associated with the condition $cond_i$ in this gateway.

In a gateway consisting of one XOR-join (Figure 3(b)), when one of incoming activities to the join is completed, the outgoing activity can start. The incoming activities need not have to be synchronized, the completion of one of the incoming activities is sufficient to trigger the beginning of the merged activity. However, we need to ensure that the follow-up activity is started only once. We capture this by the following rule.

$$transitsto(a_1, a_j) \longleftarrow xor\_join(L, a_j), happens(end(a_1)),$$
$$not\ happens(end(a_2)), not\ happens(end(a_3)), \cdots,$$
$$not\ happens(end(a_n))$$

In the above, $xor\_join(L, a_j)$ indicates that the list of activities get merged into the activity a_j. The predicate $not\ happens(end(a))$ says that the activity a has not been completed. It can be taken as negation-as-failure.

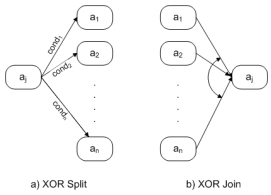

Fig. 3. (a) XOR-split and (b) XOR-join

Remark We remark that both $transitsto(.,.)$ (for all transition types) and $reversesto(.,.)$ maintain transitivity. We also assume when activity a_i finishes, activity a_j starts immediately, $happens(start(a_j)) \longleftarrow transitsto(a_i, a_j)$, which forces the rule: $happens(end(a_j)) \longleftarrow happens(start(a_j))$.

Process-flow management

A process-flow management system must be capable of both specifying and executing activities. So far, we have dealt with axioms for specifying the process flow, and describing the scheduling of pre-conditions among the activities. Now we provide the specification for execution of process flows using logic programming style of notation. We adopt a simple model.

In a process-flow management system there are many task agents that control the execution of activities. Each task agent is represented as a property in the modeled system. We assume that there is an agent (may be, process flow manager) that coordinates the execution of the activities according the specification of process flow. When an activity is considered, the manager must assign an agent that will execute the activity. Each agent can perform one or more activities; and each activity can be executed by one or more agents. When an activity is selected for execution, an agent that is qualified to perform this activity is actually chosen, and finally it is assigned this activity, if it is idle only. We use $qualified(Ag, a)$ for denoting agent Ag is qualified to activity a. We use predicate $idle(Ag)$ to denote the condition that Agent Ag is idle.

In a sequential activity (Figure 1), we use the following rule to express that an agent Ag is assigned to the activity a. This is the general rule for assigning events to agents.

$$happens(assign(Ag, a)) \longleftarrow holds\_at(idle(Ag)),$$
$$holds\_at(waiting(a, Ag)), happens(release(Ag)),$$
$$qualified(Ag, a)$$

Let us now consider the following triggering rules. When an agent is assigned to an activity, the activity is triggered. This is captured as,

$$happens(begin(Ag, a)) \longleftarrow happens(assign(Ag, a)),$$
$$happens(start(a)).$$

When an activity is being executed by an agent, the agent is not idle any more, until the activity is completed. Once the activity finishes, the agent is released, and becomes idle, and ready to execute the next activity. Let us formulate two rules.

$$happens(over(Ag, a)) \longleftarrow happens(end(a)), \\ happens(begin(Ag, a))$$
$$happens(release(Ag, a)) \longleftarrow happens(over(Ag, a))$$

We need some initial conditions also. We shall assume that there will be some designated event e which will start the execution of the process. The corresponding initial condition can be: $initiates(e, idle(Ag))$, where we assume all the agents are idle.

V. Capturing UBL process in OWL

In this section we take an example of a simple UBL process and describe how we can specify it using a logical framework. We consider a fragment of first-order logic, which is a semantic markup language called OWL (Web Ontology Language) [3], proposed by W3C Web Ontology Working Group. The primary reason for choosing this logic is that it is decidable and a popular language for developing ontologies.

OWL with Rules (SWRL)

OWL is a set of XML elements and attributes, with standardized meaning, that are used to define terms and their relationships. It is possible to declare classes and organize them hierarchically in a subsumption relation in this framework. OWL allows combining classes using intersection, union or complementation, or as enumerations of specified objects. The domains of OWL properties are classes, and ranges can be either OWL classes (if they are object properties), or externally-defined data-types such as string, or integer (if these are data-type properties). OWL also permits to make a property to be transitive, symmetric, functional, or inverse of another property. OWL can express which objects (also called "individuals") belong to which classes and what the property values are of specific individuals. One can make equivalent statements on classes and on properties, disjointness statement on classes, and assert equality and inequality between individuals. While the syntax of OWL corresponds to that of RDF the semantics of OWL are extensions of the semantics of Description Logic. Thus, OWL shares many common features with Description Logic (DL) [18], [1]. OWL still has limited expressivity, in particular, it cannot express much about the properties. In order to overcome these limitations Horrocs et. al. have extended OWL with Horn Clause rules in a syntactically and semantically coherent manner, which is called Semantic Web Rules Language (SWRL) [11]. The basic syntax of SWRL rules is an extension of the OWL syntax, and SWRL rules are interpreted by extending the model theoretic semantics for OWL. However an arbitrary extension of OWL with rules could easily lead to undecidability of interesting problems. For this reason, we restrict ourselves to a decidable fragment of such extension, by using the so-called DL-safe rules [13], which forces each variable in a rule to occur in a non-DL atom in the rule body.

We employ a two-step technique for specifying a UBL process details, and thus generating ontologies in the end.

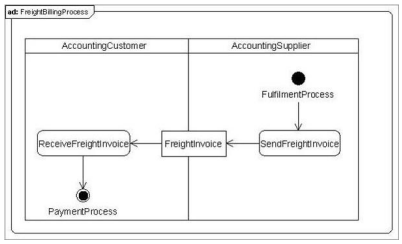

Fig. 4. Activity Diagram for Freight Billing Process

In the first step, we specify the process flow in OWL, followed by associating semantics with UBL documents in OWL.

A Case study

Let us now describe a case study of a UBL process. Here, all the agents involved, are added to the ontology as classes. The agents are linked using object-properties based on the subprocesses through which they interact. The domain and range for these properties are the classes for the agents and objects that are involved in that subprocess. One can also define the inverse properties for the corresponding object-properties, wherever they would be useful for querying purposes. As an example, let us consider a simple process - Freight Billing process; given in Figure 4, which is bereft of any gateways.

In this process diagram, while *SendFreightInvoice* and *ReceiveFreightInvoice* are Activities, *FreightInvoice* is an Object. *FulfilmentProcess* generates an Initial Node (given by a filled circle), and *PaymentProcess* a Final Node (a Bull's eye). Correspondingly, we add activities, $Initial\_activity, Final\_activity$, $doFulfilmentProcess, doPaymentProcess$ etc. We introduce a class *Events* for events. We assume each activity occurs instantaneously. For each activity a, we generate an event $event\_a$ to denote its occurrence. We consider only three sequential transitions here,

$Initial\_activity \rightarrow SendFreightInvoice$,
$SendFreightInvoice \rightarrow ReceiveFreightInvoice$,
$ReceiveFreightInvoice \rightarrow Final\_activity$.

AccountingCustomer and *AccountingSupplier* represent two Partitions (swim lanes), which dictates the assignment of agents to events. We consider three Agents/Actors - System, AccountingCustomer, and AccountingSupplier.

Equivalent OWL Ontology for the Freight Billing process could be constructed as follows. *Agents* and *UBLDocuments* are designed as OWL Classes of which the agents involved and the UBL documents involved (given by Objects) become sub-classes, respectively. *System*, *AccountingCustomer* and *AccountingSupplier*, thus become sub-classes of Class *Agents*, and *FreightInvoice* becomes a sub-class of Class *UBLDocuments*. *Initial_activity*, *SendFreightInvoice*, *ReceiveFreightInovice*, *Final_activity*, *doFulfilmentProcess* and *doPaymentProcess* become subclasses of Class *Activities*.

We state some ObjectProperties that we would be using to generate ontologies. We shall use $relDocuments \subseteq Activities \times UBLDocuments$ to relate activities with their respective UBLDocuments. Similarly $follows \subseteq Activities \times Activities$ to show the sequence of actions. Accordingly, we can instantiate these roles with respect to this FrieghtBilling Process corresponding to three transitions given above.

Let us now capture different stages of the process with rules. We introduce a class *happens* as a subclass of Events. Consequently, $happens(event\_a)$ means that activity a has occurred. Similarly, we introduce a class called *Conditions*, and *holds_at* as a subclass of it. This class can be instantiated with names $Initiation, Closure$ etc, where $Initiation$ is a condition for the FrieghtBilling Process to begin, and $Closure$ is a condition for the FrieghtBilling Process to end. Then there is the role *initiates* whose domain is Events, and range is the class Conditions. Similarly, is the role *terminates*.

For pre-condition we need two facts: $initiates(event\_doFulfilmentProcess, Initiation)$, $happens(event\_doFulfilmentProcess)$. We also need the following rule: $happens(event\_Initial\_activity) \longleftarrow holds\_at(Initiation)$.

Similarly, the post-condition requires the fact, $terminates(event\_Final\_activity, Closure)$. We need the following rule too: $happens(event\_PaymentProcess) \longleftarrow holds\_at(Closure)$.

We consider the sequential activities discussed earlier, and write the *happens* rules corresponding to sequential transitions, *e.g.*, for the transition, $follows(Initial\_activity, SendFrieghtInvoice)$ we generate the rule:

$$happens(event\_SendFrieghtInvoice) \longleftarrow follows(Initial\_activity, SendFrieghtInvoice), happens(event\_Initial\_activity)$$

Let us now discuss the assignment of agents as part of Frieghtbilling process-flow management. We include additional conditions, *idle_Ag* for all agents: $System, AccountingSupplier$, and $AccountingCustomer$. We need the initial conditions as $initiates(event\_doFulfilmentProcess, idle - Ag)$. Using the axiom for $holds\_at(.)$ we can conclude $holds\_at(idle\_Ag)$ is true taking into account $happens(event\_doFulfilmentProcess)$. Also we use the following facts to denote that a particular agent is qualified to perform a particular activity.

$qualified(System, Initial\_Activity)$;
$qualified(AccountingSupplier, SendFrieghtInvoice)$;
$qualified(AccountingCustomer, ReceiveFrieghtInvoice)$;
$qualified(System, Final\_Activity)$.

For using OWL we need to introduce the following roles, $assign, begin, over \subseteq Agent \times Activities$, and formulate the following rules:

$assign(Ag, a) \longleftarrow holds\_at(idle\_Ag), release(Ag, a)$, $qualified(Ag, a)$;
$begin(Ag, a) \longleftarrow assign(Ag, a), happens(event\_a)$;
$over(Ag, a) \longleftarrow begin(Ag, a), happens(event\_a)$;
$release(Ag, a) \longleftarrow over(Ag, a)$, etc.

Finally, we instantiate the rules for assigning events to agents, and triggering events as well. We consider the XMI representation of this UBL process diagrams and write an XSLT to generate OWL ontologies along with rules. A preliminary version of this mapping appeared in [17].

V. Specifying UBL documents with OWL

In the second stage we generate OWL ontology out of XSD schema. A major difficulty in this task is the difference between semantics of XSD and OWL documents, - while XSD has a tree-like model, OWL semantics is firmly entrenched in logical foundations. This has to be partially overcome by establishing suitable mappings between the different data model elements of XSD and OWL. A model ontology is generated out of XSD schema automatically using techniques from [6].

In order to support the representation of domain knowledge, the extracted model of OWL out of XML Schemas consists of three parts: classes to define concepts, object properties to relate different objects together, and datatype properties to relate objects to data type values. For handling xsd:complexTypes and xsd:elements we generate classes according to rules as follows. An element in the source XML tree, being a leaf containing only a literal or an attribute, is always mapped to a owl:DatatypeProperty having the surrounding class as domain. For any other sub-element, it would be mapped to an owl:ObjectProperty with the source element as the domain and the sub-element as the range. Actually for nested elements, when one element contains another element (neither a literal nor an attribute) we assume a "part-of" relationship. This corresponds to an owl:ObjectProperty, which establishes a relationship between two classes.

XSD schema also contain arity constraints like xsd:minOccurs and xsd:maxOccurs which are mapped to equivalent cardinality constraints owl:minCardinality and owl:maxcardinality respectively in OWL. For elements appearing in xsd:sequence and xsd:all, they are clubbed into the complex class owl:intersectionOf. We summarize our mapping in Table I. We have used Perl for conversion purposes.

XSD Schema Elements	OWL Elements
xsd:elements, containing other elements or having at least one attribute	owl:Class, coupled with owl:ObjectProperties
xsd:elements, with neither sub-elements nor attributes	owl:DatatypeProperties
named xsd:ComplexType	owl:Class
named xsd:SimpleType	owl:DatatypeProperties
xsd:minOccurs, xsd:maxOccurs	owl:minCardinality, owl:maxCardinality
xsd:sequence, xsd:all	owl:intersectionOf

TABLE I
MAPPING BETWEEN ELEMENTS OF XSD SCHEMAS AND OWL

VI. Query Development

For querying the OWL ontology thus developed we make use of SQWRL (Semantic Query-Enhanced Web Rule Language), a SWRL-based query language [15]. SQWRL provides SQL-like operations that can be used to format knowledge retrieved from an OWL ontology. SQWRL closely follows SWRL's semantics and uses the standard SWRL presentation syntax supported by the SWRLTab [10]. SQWRL is defined using a library of SWRL built-ins, and it effectively turns SWRL [11], [15] into a query language.

We provide a query catalog of queries that can be posed on the ontology created from both UBL Documents and processes in Table II.

Query Statement on UBL documents
Q1. Find all the instance documents and their corresponding ID's **Q2**. Find all the instances, issue-dates and IDs such that the issue date and ID correspond to the same instance **Q3**. Count the number of documents with the same issue dates **Q4**. Sort the documents according to their issue dates **Q5**. Given the value of ID (which is unique for each document), obtain the values of other elements of that instance document **Q6**. Get the ID of the Instance documents along with the Customer Name and the Supplier Name
Query Statement on UBL process diagrams
Q1. End of which process initiates the Freight Billing Process? **Q2**. Which process starts at the end of Freight Billing Process? **Q3**. Which Activities follow the Activity "SendFreightInvoice"? **Q4**. Which Activities are related to which UBL Documents? **Q5**. Which Agent is qualiffied to perform which Activities?

TABLE II
QUERIES ON UBL PROCESS

VII. Conclusion

We have created ontologies out of UBL processes using OWL as there is a need for a common standard framework to store information, manage data, and retrieve meaningful information out of it. The next step is to appropriately tune this mapping so that more complex constructs like, fork, join, split, merge etc. can be accommodated for specification in OWL.

An application of this kind of ontology can be seen in requirements authoring. The aim of such a work should be to build a system which can detect incompleteness and inconsistency in requirements. We plan to use ontology created from the artifacts to constraint the authoring of requirements so that we can minimize the possibility of requirements deviations and inconsistencies. For example, let us consider a software requirements specification (SRS) for Freightbilling process, which a set of *use cases* that describe all of the interactions that the users will have with the software. We present a fragment of such use cases.

Preconditions: End of fulfilment process.
Actors/Agents: System, Accounting Supplier, Accounting Customer.
Business Trigger: Initiation of the freight billing upon completion of the fulfilment process.
Execution: The invoice is submitted to the Accounting Customer by the Accounting Supplier, the Accounting Customer receives the invoice etc.

For all such use cases we can formulate suitable OWL/SWRL formulas and check their consistency against the generated ontology. Inconsistency indicates that there is a divergence between the textual specifications and the process models. If this checking done at a greater frequency, and with respect to smaller chunks of text, this can provide a basis for near real-time constrained requirements authoring.

References

[1] F. Baader and W. Nutt. Basic Description Logics. In *Description Logic Handbook*, pages 43–95, 2003.
[2] S. Battle. Round-tripping between XML and RDF. In *In International Semantic Web Conference (ISWC)*, Hiroshima, Japan, 2004. Springer.
[3] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. *OWL Web Ontology Language Reference*. W3C Recommendation, 2003.
[4] T. Berners-Lee, J. Hendler, and O. Lassila. *The semantic web*. Scientific American, 2004.
[5] C. Bettini, X. Wang, and S. Jajodia. Temporal reasoning in workflow systems, distributed and parallel databases. *Distributed and Parallel Databases*, 11(3):269306, 2002.
[6] H. Bohring and S. Auer. Mapping XML to OWL Ontologies. In *Leipziger Informatik-Tage, LNI*, volume 72, pages 147–156. GI, 2005.
[7] N. K. Cicekli and I. Cicekli. Formalizing the specification and execution of workflows using the Event Calculus. *Information Sciences*, 176(15):2227–2267, 2006.
[8] H. Davulcu, M. Kifer, C. R. Ramakrishnan, and I. Ramakrishnan. Logic based modeling and analysis of workflows. In *In Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–3. ACM Press, 1998.
[9] A. Geppert, M. Kradolfer, and D. Tombros. Realization of cooperative agents using an active object-oriented database system. In *The Second International Workshop on Rules in Database Systems (RIDS)*, pages 327–341, Athens, Greece, 1995.
[10] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe. *A Practical Guide To Building OWL Ontologies Using The Protege-OWL Plugin and CO-ODE Tools Edition 1.0*, 2004. http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf.
[11] I. Horrocks, P. F. Patel-Schneider, S. Bechhofer, and D. Tsarkov. OWL rules: A proposal and prototype implementation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):23–40, July 2005.
[12] P. Koksal, N. Cicekli, and H. Toroslu. Specification of workflow processes using the action description language c. In *AAAI Spring 2001 Symposium Series: Answer Set Programming*, pages 103–109, Palo Alto, California, 2001.
[13] B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. In *Journal of Web Semantics*, pages 549–563. Springer, 2004.
[14] *OASIS*. available at http://www.oasis-open.org/home/index-php.
[15] M. O'connor, S. Tu, C. Nyulas, A. Das, and M. Musen. Querying the Semantic Web with SWRL. *Advances in Rule Interchange and Applications*, pages 155–159, 2007.
[16] F. Puhlmann and M. Weske. Using the π-calculus for formalizing workflow patterns. In *Business Process Management*, pages 153,168, 2005.
[17] K. Sawant and S. Roy. A mapping of UBL process diagrams to OWL. As a poster paper in 3rd Indian Software Engineering Conference, 2010. Mysore.
[18] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
[19] W. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
[20] W3C Recommendation. *Universal Business Language v2.0*, 2006. available at http://docs.oasis-open.org/ubl/cs-UBL-2.0/UBL-2.0.html.